CMSC 22610 Winter 2004	Implementation of Computer Languages	Project 2 January 21, 2004

Mini-Lua parser

Due: February 4, 2004

1 Introduction

Your second assignment is to implement a parser for Mini-Lua. You will use ML-Yacc to generate a parser from an LALR(1) specification (see Chapter 3 of Appel's book). The actions of this parser will construct a *parse tree* representation for a Mini-Lua program. In addition to writing the parser, you will also be responsible for defining the SML datatypes that represent the parse tree.

2 The Mini-Lua grammar

The concrete syntax of Mini-Lua is specified by the grammar given in Figures 1 and 2. To make this grammar unambiguous, the precedence of operators must be specified. They are (from weakest to strongest):

```
or
and
< > <= >= ~= ==
..
+ -
* /
not - (unary)
^
```

All binary operators, except "`..`" (concatenation) and "`^`" (exponentiation), are left associative.

3 Requirements

Your implementation should consist of the following five files:

`mini-lua.cm` — a CM sources file for compiling your project.

`main.sml` — An SML source file containing the definition a structure `Main`, that defines a function

Block
::= (*Stmt* **;**)*

Stmt
::= *Vars* **=** *Exps*
| *FunctionCall*
| **do** *Block* **end**
| **while** *Exp* **do** *Block* **end**
| **if** *Exp* **then** *Block* (**elseif** *Exp* **then** *Block*)* (**else** *Block*)opt **end**
| **return** *Exps*opt
| **break**
| **for** Name **=** *Exp* **,** *Exp* (**,** *Exp*)opt **do** *Block* **end**
| **for** *Names* **in** *Exps* **do** *Block* **end**
| **local**opt **function** Name *FunctionBody*
| **local** *Names* **=** *Exps*

Exps
::= *Exp* (**,** *Exp*)*

Exp
::= *Exp BinOp Exp*
| **not** *Exp*
| **-** *Exp*
| *PrefixExp*
| *Function*
| **{** (*Field* (**,** *Field*)*)opt **}**
| **nil**
| **true**
| **false**
| Number
| String

PrefixExp
::= *Var*
| *FunctionCall*
| **(** *Exp* **)**

Field
::= **[** *Exp* **]** **=** *Exp*
| Name **=** *Exp*

BinOp
::= **or** | **and** | **<** | **>** | **<=** | **>=** | **~=** | **==** | **..** | **+** | **-** | ***** | **/** | **^**

Figure 1: The concrete syntax of Mini-Lua (A)

Vars
::= *Var* (**,** *Var*)*

Var
::= Name
| *PrefixExp* **[** *Exp* **]**
| *PrefixExp* **.** Name

Function
::= **function** *FunctionBody*

FunctionBody
::= **(** *Params*^{opt} **)** *Block* **end**

FunctionCall
::= *PrefixExp Args*
| *PrefixExp* **:** Name *Args*

Args
::= **(** *Exps*^{opt} **)**
| **{** (*Field* (**,** *Field*)*)^{opt} **}**

Figure 2: The concrete syntax of Mini-Lua (B)

```
val parseFile : string -> LuaParseTree.program
```

where `LuaParseTree.program` is the type of program parse trees. This function should open the named source file, parse it, and return the resulting tree.

`lua-parse-tree.sml` — An SML file containing a module `LuaParseTree` that defines the parse-tree representation of Mini-Lua programs.

`lua.y` — An ML-Yacc specification file for parsing Mini-Lua programs. The actions of this parser should construct parse tree nodes.

`lua.l` — An ML-Lex specification file for lexing Mini-Lua. We will provide an skeleton for this file. You may also choose to use a modified version of the lexer you wrote for Part 1 of the project.

4 Document history

Jan. 30 Added missing grammar rules for functions and function calls.

Jan. 27 Added missing grammar rules for variables.

Jan. 21 Original version.